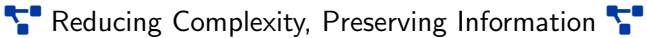
Unsupervised Learning

Dimensionality Reduction: Principal Component Analysis (PCA) & t-SNE Basics

Sarwan Ali

Department of Computer Science
Georgia State University



Today's Learning Journey

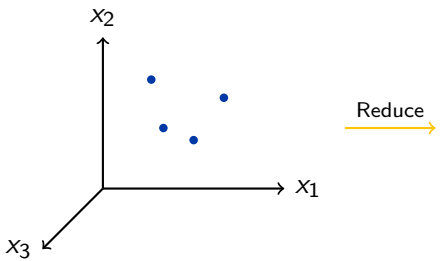
1. Introduction to Dimensionality Reduction
2. Principal Component Analysis (PCA)
3. t-SNE Basics
4. Practical Implementation
5. Advanced Topics and Extensions
6. Summary and Best Practices

What is Dimensionality Reduction?

Definition: Process of reducing the number of features (dimensions) in a dataset while preserving important information.

Goal: Transform high-dimensional data to lower-dimensional space

- Reduce computational complexity
- Remove noise and redundancy
- Enable visualization
- Overcome curse of dimensionality



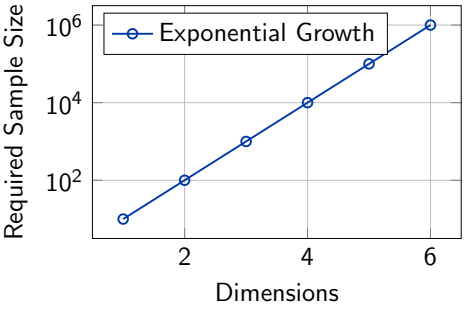
3D Data

The Curse of Dimensionality

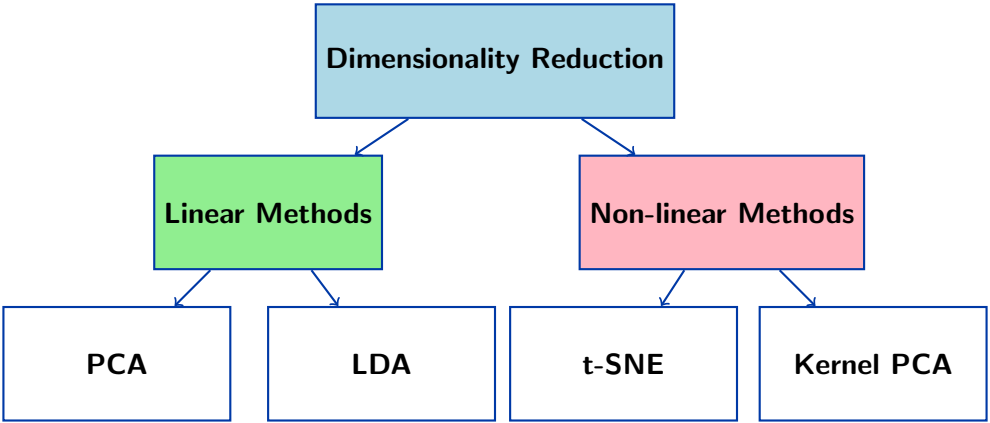
Problems with High Dimensions:

- Exponential growth of data space
- Distances become less meaningful
- Sparse data distribution
- Computational complexity increases
- Visualization becomes impossible

Example: In a 10-dimensional unit hypercube, 99.9% of the volume is within 0.05 of the surface!



Types of Dimensionality Reduction



Today's Focus: Principal Component Analysis (PCA) and t-SNE basics

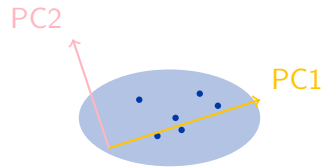
What is Principal Component Analysis?

Definition

PCA is a linear dimensionality reduction technique that transforms data to a lower-dimensional space by finding the directions (principal components) of maximum variance.

Key Ideas:

- Find directions of maximum variance
- Orthogonal principal components
- Preserve most important information
- Linear transformation



Data with Principal Components

Mathematical Foundation of PCA

Step 1: Data Preparation

$$\text{Given data matrix: } \mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n]^T \in \mathbb{R}^{n \times d} \tag{1}$$

$$\text{Center the data: } \tilde{\mathbf{X}} = \mathbf{X} - \boldsymbol{\mu} \text{ where } \boldsymbol{\mu} = \frac{1}{n} \sum_{i=1}^{n} \mathbf{x}_i \tag{2}$$

Step 2: Covariance Matrix

$$\mathbf{C} = \frac{1}{n-1} \tilde{\mathbf{X}}^T \tilde{\mathbf{X}} \in \mathbb{R}^{d \times d} \tag{3}$$

Step 3: Eigendecomposition

$$\mathbf{C}\mathbf{v}_i = \lambda_i \mathbf{v}_i \text{ for } i = 1, 2, \ldots, d \tag{4}$$

$$\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_d \geq 0 \tag{5}$$

where $\mathbf{v}_i$ are eigenvectors (principal components) and λ_i are eigenvalues

PCA Algorithm Steps

1. **Standardize/Center** the data

$$\tilde{x}_{ij} = \frac{x_{ij} - \mu_j}{\sigma_j} \quad \text{or} \quad \tilde{x}_{ij} = x_{ij} - \mu_j \tag{6}$$

2. **Compute covariance matrix**

$$\mathbf{C} = \frac{1}{n-1}\tilde{\mathbf{X}}^T\tilde{\mathbf{X}} \tag{7}$$

3. **Find eigenvalues and eigenvectors**

$$\mathbf{C} = \mathbf{V}\mathbf{\Lambda}\mathbf{V}^T \tag{8}$$

4. **Select top k components**

$$\mathbf{W} = [\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_k] \in \mathbb{R}^{d\times k} \tag{9}$$

5. **Transform data**

$$\mathbf{Y} = \tilde{\mathbf{X}}\mathbf{W} \in \mathbb{R}^{n\times k} \tag{10}$$

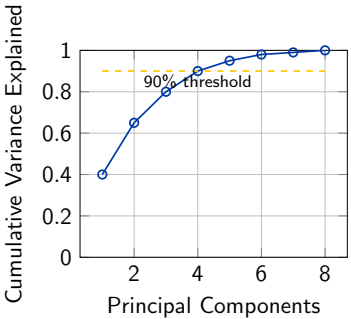
Choosing Number of Components

Explained Variance Ratio:

$$\text{EVR}_i = \frac{\lambda_i}{\sum_{j=1}^{d} \lambda_j} \tag{11}$$

Cumulative Explained Variance:

$$\text{CEV}_k = \frac{\sum_{i=1}^{k} \lambda_i}{\sum_{j=1}^{d} \lambda_j} \tag{12}$$

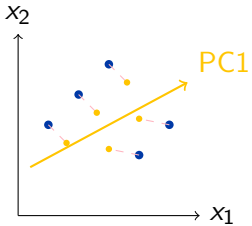


Common Threshold: Retain components explaining 90-95% of variance

Elbow Method: Look for the "elbow" in the scree plot where eigenvalues drop significantly

PCA Example: 2D to 1D

Original Data:



Mathematics:

$$\text{Data: } \mathbf{X} = \begin{bmatrix} 0.5 & 1.5 \\ 1.0 & 2.0 \\ 1.5 & 2.5 \\ 2.0 & 1.0 \\ 2.5 & 1.5 \end{bmatrix} \tag{13}$$

$$\text{Centered: } \tilde{\mathbf{X}} = \mathbf{X} - \boldsymbol{\mu} \tag{14}$$

$$\text{Covariance: } \mathbf{C} = \begin{bmatrix} 0.75 & 0.25 \\ 0.25 & 0.35 \end{bmatrix} \tag{15}$$

$$\text{Eigenvalues: } \lambda_1 = 0.9, \lambda_2 = 0.2 \tag{16}$$

$$\text{PC1: } \mathbf{v}_1 = [0.8, 0.6]^T \tag{17}$$

Advantages and Limitations of PCA

Advantages:

- Reduces computational complexity
- Removes correlated features
- No hyperparameters to tune
- Mathematically well-founded
- Reversible transformation
- Works well for linear relationships

Limitations:

- Assumes linear relationships
- Principal components may be hard to interpret
- Sensitive to scaling
- May not preserve local structure
- Not suitable for sparse data
- Global method (considers all data)

When to Use PCA

- High-dimensional data with linear correlations
- Need to reduce noise and computational cost
- Want to visualize high-dimensional data
- Preprocessing for other ML algorithms

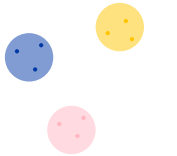
Introduction to t-SNE

t-Distributed Stochastic Neighbor Embedding (t-SNE)

A non-linear dimensionality reduction technique particularly well-suited for visualization of high-dimensional datasets by preserving local structure.

Key Concepts:

- Preserves local neighborhood structure
- Uses probability distributions
- Non-linear and non-parametric
- Excellent for visualization
- Stochastic optimization



Developed by: Laurens van der Maaten and Geoffrey Hinton (2008)

t-SNE Algorithm Overview

Main Idea: Convert similarities between data points to joint probabilities and minimize KL-divergence between high and low-dimensional representations.

Step 1: Compute pairwise similarities in high-dimensional space

$$p_{j|i} = \frac{\exp(-\|\mathbf{x}_i - \mathbf{x}_j\|^2/2\sigma_i^2)}{\sum_{k\neq i}\exp(-\|\mathbf{x}_i - \mathbf{x}_k\|^2/2\sigma_i^2)} \tag{18}$$

Step 2: Symmetrize probabilities

$$p_{ij} = \frac{p_{j|i} + p_{i|j}}{2n} \tag{19}$$

Step 3: Compute similarities in low-dimensional space using t-distribution

$$q_{ij} = \frac{(1 + \|\mathbf{y}_i - \mathbf{y}_j\|^2)^{-1}}{\sum_{k\neq l}(1 + \|\mathbf{y}_k - \mathbf{y}_l\|^2)^{-1}} \tag{20}$$

t-SNE Optimization

Step 4: Minimize KL-divergence between P and Q

$$C = KL(P||Q) = \sum_{i,j} p_{ij} \log \frac{p_{ij}}{q_{ij}} \tag{21}$$

Gradient:

$$\frac{\partial C}{\partial \mathbf{y}_i} = 4 \sum_j (p_{ij} - q_{ij})(\mathbf{y}_i - \mathbf{y}_j)(1 + \|\mathbf{y}_i - \mathbf{y}_j\|^2)^{-1} \tag{22}$$

Key Hyperparameters:

- **Perplexity:** Controls local neighborhood size (5-50)
- **Learning rate:** Step size (10-1000)
- **Iterations:** Number of optimization steps (1000+)

Why t-distribution?

- Heavy tails
- Prevents crowding
- Better separation in low dimensions

t-SNE vs PCA Comparison

Aspect	PCA	t-SNE
Method	Linear transformation	Non-linear embedding
Preserves	Global structure, variance	Local structure, neighborhoods
Computational Cost	$O(nd^2)$	$O(n^2)$
Deterministic	Yes	No (stochastic)
Hyperparameters	None	Perplexity, learning rate
Interpretability	Moderate	Low
New Data	Easy projection	Difficult
Best Use	Preprocessing, noise reduction	Visualization, exploration

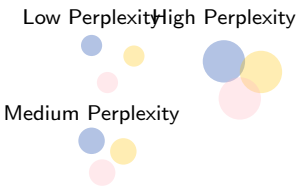
Recommendation: Use PCA for dimensionality reduction in ML pipelines, t-SNE for data visualization and exploration.

t-SNE Hyperparameter: Perplexity

Perplexity measures the effective number of local neighbors.

$$\text{Perplexity}(P_i) = 2^{H(P_i)} \tag{23}$$

where $H(P_i) = -\sum_j p_{j|i} \log_2 p_{j|i}$



Effects of Perplexity:

- **Low (5-15):** Focus on very local structure
- **Medium (20-50):** Balance local/global
- **High (50+):** More global structure

Rule of thumb: Perplexity should be smaller than the number of points, typically between 5-50.

Common Issues and Solutions in t-SNE

Common Issues:

- Crowding problem
- Curse of intrinsic dimension
- Non-deterministic results
- Sensitive to hyperparameters
- Computational complexity
- Difficult interpretation of distances

Solutions:

- Use t-distribution (heavy tails)
- PCA preprocessing for high dimensions
- Run multiple times with different seeds
- Tune perplexity systematically
- Use early exaggeration
- Focus on cluster patterns, not distances

Best Practices

- Apply PCA first if $d > 50$
- Try perplexity values: 5, 30, 50, 100
- Run for sufficient iterations (≥ 1000)
- Check convergence of cost function

PCA Implementation in Python

```python
import numpy as np
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt

# Load and prepare data
X = ... # Your high-dimensional data
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Apply PCA
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)

# Analyze results
print(f"Explained variance ratio:
      {pca.explained_variance_ratio_}")
print(f"Cumulative variance:
      {np.cumsum(pca.explained_variance_ratio_)}")

# Visualize
plt.scatter(X_pca[:, 0], X_pca[:, 1])
plt.xlabel('First Principal Component')
plt.ylabel('Second Principal Component')
plt.show()
```

Key Parameters:

- `n_components`: Number of components
- `whiten`: Normalize components
- `svd_solver`: Algorithm choice

Useful Attributes:

- `components_`: Principal axes
- `explained_variance_`: Eigenvalues
- `mean_`: Per-feature mean

t-SNE Implementation in Python

```python
import numpy as np
from sklearn.manifold import TSNE
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt

# Prepare data (apply PCA first if high-dim)
if X.shape[1] > 50:
    pca = PCA(n_components=50)
    X_reduced = pca.fit_transform(X)
else:
    X_reduced = X

# Apply t-SNE with different perplexities
perplexities = [5, 30, 50]
fig, axes = plt.subplots(1, 3, figsize=(15, 5))

for i, perp in enumerate(perplexities):
    tsne = TSNE(n_components=2,
                perplexity=perp,
                random_state=42,
                n_iter=1000)
    X_tsne = tsne.fit_transform(X_reduced)

    axes[i].scatter(X_tsne[:, 0], X_tsne[:, 1],
                    c=y, cmap='tab10')
    axes[i].set_title(f'Perplexity = {perp}')
plt.show()
```

Key Parameters:

- `perplexity`: 5-50 typically
- `learning_rate`: 10-1000
- `n_iter`: ≥ 1000
- `early_exaggeration`: 12.0
- `random_state`: For reproducibility

Tips:

- Monitor `kl_divergence_`
- Use `init='pca'` for better initialization
- Consider `method='barnes_hut'` for speed

Case Study: Iris Dataset Visualization

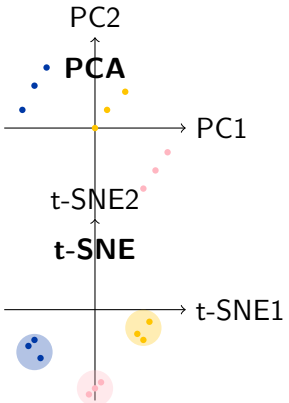
Dataset: 150 samples, 4 features, 3 classes

PCA Results:

- PC1: 72.8% variance
- PC2: 22.9% variance
- Total: 95.7% with 2 components
- Clear linear separation

t-SNE Results:

- Better cluster separation
- Non-linear boundaries preserved
- Perplexity = 30 works well
- More compact clusters



Variants and Extensions

PCA Variants:

- **Kernel PCA:** Non-linear extension using kernel trick
- **Sparse PCA:** Interpretable components with sparsity
- **Incremental PCA:** For streaming/large datasets
- **Probabilistic PCA:** Bayesian approach
- **Independent Component Analysis (ICA):** Statistical independence

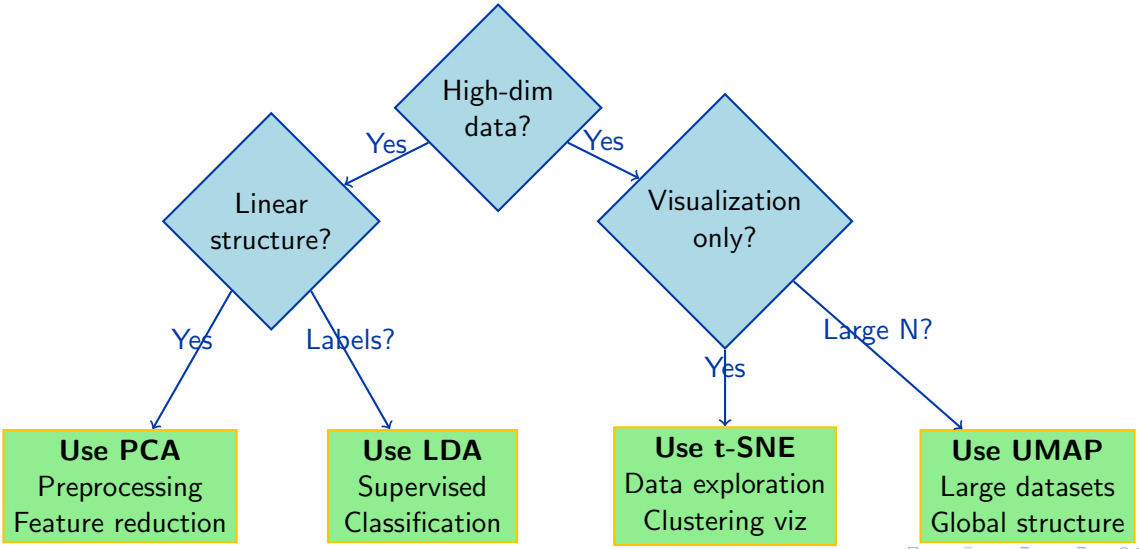
t-SNE Extensions:

- **Barnes-Hut t-SNE:** $O(N \log N)$ complexity
- **Parametric t-SNE:** Neural network-based
- **UMAP:** Uniform Manifold Approximation
- **LargeVis:** For very large datasets
- **Multi-scale SNE:** Multiple perplexities

Modern Alternatives

- **UMAP:** Faster than t-SNE, preserves global structure better
- **Autoencoders:** Deep learning approach to dimensionality reduction
- **Variational Autoencoders (VAE):** Probabilistic latent representations

When to Use Which Method?



Evaluation Metrics for Dimensionality Reduction

Quantitative Metrics:

- **Reconstruction Error:**

$$E = \|\mathbf{X} - \hat{\mathbf{X}}\|_F^2 \tag{24}$$

- **Explained Variance Ratio:**

$$EVR = \frac{\text{Var}(\mathbf{Y})}{\text{Var}(\mathbf{X})} \tag{25}$$

- **Trustworthiness:** Local neighborhood preservation
- **Continuity:** Smooth mapping preservation

Qualitative Assessment:

- Visual cluster separation
- Preservation of known structure
- Interpretability of components
- Stability across runs

Task-specific Evaluation:

- Classification accuracy after reduction
- Clustering quality metrics
- Downstream task performance

Key Takeaways

Principal Component Analysis (PCA)

- **Best for:** Linear relationships, preprocessing, noise reduction
- **Strengths:** Fast, deterministic, interpretable, reversible
- **Limitations:** Linear assumptions, global method

t-Distributed Stochastic Neighbor Embedding (t-SNE)

- **Best for:** Visualization, cluster exploration, non-linear data
- **Strengths:** Preserves local structure, handles non-linearity
- **Limitations:** Stochastic, hyperparameter sensitive, not for new data

General Principle

Choose dimensionality reduction technique based on your specific goal: preprocessing (PCA), visualization (t-SNE), or both in combination.

Best Practices and Recommendations

Data Preprocessing:

- Always center/standardize data for PCA
- Handle missing values appropriately
- Consider outlier detection
- Apply PCA before t-SNE for high dimensions

Parameter Selection:

- PCA: Use scree plot or cumulative variance
- t-SNE: Try multiple perplexity values
- Validate with domain knowledge

Practical Workflow:

1. Explore data with PCA first
2. Choose number of components wisely
3. Use t-SNE for final visualization
4. Validate results with known structure
5. Document hyperparameters used

Common Pitfalls:

- Over-interpreting t-SNE distances
- Using too few iterations
- Ignoring computational constraints

Next Steps and Further Reading

Advanced Topics to Explore:

- Manifold learning theory
- Autoencoders for dimensionality reduction
- UMAP and other modern methods
- High-dimensional data analysis
- Streaming dimensionality reduction

Practice Exercises:

- Apply PCA to image datasets
- Compare t-SNE with different perplexities
- Implement PCA from scratch
- Evaluate reconstruction quality

Recommended Resources:

- van der Maaten & Hinton (2008) - Original t-SNE paper
- "Pattern Recognition and Machine Learning" - Bishop
- Scikit-learn documentation
- "Dimensionality Reduction: A Comparative Review" - van der Maaten et al.

Tools and Libraries:

- `sklearn.decomposition`
- `sklearn.manifold`
- `umap-learn`
- `plotly` for interactive visualization

Thank You!

Questions & Discussion

💡 Remember: The best dimensionality reduction technique depends on your data and goals!

Next: Association Rules: Market basket analysis, Apriori algorithm